Hardware enclaves & oblivious computation

Slides adapted from here
Hardware enclaves

• An alternative way to compute on encrypted data

• Hardware abstractions for distributing trusted execution to untrusted platforms
Real world threats to trusted execution

- Malicious software
  - Rootkits in OS, malicious kernel

- Cold-boot attacks
  - Memory modules do not immediately lose data after loss of power
  - Attacker with physical access can perform a memory dump of a machine’s RAM by abruptly rebooting a target machine and then booting a re-installed OS from a flash drive
  - Literally **cold** as liquid nitrogen can be used to prolong data remanence
  - Software-based disk encryption can be circumvented
Hardware enclaves architecture

Attacker can compromise almost the entire server-side software stack
Trusted CPU

- Content is stored unencrypted in registers and cache (cannot be read by the adversary)

- Adversary cannot change enclave program execution

  - Any interruption/exception triggers an asynchronous exit (AEX) operation

  - Enclave context is saved in the EPC, registers are erased, and control flow is returned to the external program
Enclave page cache

• Enclave pages are encrypted and stored in Enclave Page Cache (EPC)
• All enclaves share this space, but pages are encrypted under different keys
• Memory encryption engine (MEE) encrypts all evicted data from the cache
• EPC pages can also be evicted to main memory by the OS
• Content integrity is protected by MACs over pages plus a Merkle tree
• EPC pages used to be limited to 93.5 MB
  • Recent Icelake SGX has up to 1 TB of EPC!
Remote attestation

- Each enclave has a set of unique keys that are generated by a root secret embedded in the trusted CPU hardware

- Enclave produces a signed message, including a *measurement* that identifies the loaded program
  
  - Group signature scheme that preserves anonymity of the signers

- The signature can be verified by a remote user using the trusted Intel Attestation Service (IAS)
  
  - Group public key used to verify enclave is genuine
Enclaves are widely available in the cloud

- Azure supports SGX
- GCP supports AMD SEV (Secure Encrypted Virtualization)
- AWS supports Nitro (hypervisor-based)

Enclaves are prone to side channel attacks on encrypted data
Memory side channels

- OS is still in charge of resource management
- OS controls page table (mapping between virtual pages and physical pages)
  - Can reclaim physical pages (swap page to disk) and restore page mappings (load physical page from disk)
  - OS must know the virtual base address of the page at which the page fault occurred (though not within a page)
- Controlled-leakage channel attack shows how the OS can trigger page faults
  - Extract text documents from font rendering engine & spell checker
  - Obtain outlines of JPEG images decompressed by libjpeg
Network side channels

• SGX only meant to handle single machine applications

• Distributed applications need network communication

• A powerful attacker can analyze traffic patterns even if communication is encrypted

• This paper attacks VC3 (a MapReduce system on SGX) by observing volume of encrypted communication
How to protect against access pattern leakage?

• Regular computation leaks due to *data-dependent access patterns*

• Access patterns will change depending on the data content, revealing information even if all data content is encrypted

• *Oblivious algorithms* can be used to protect against such leakage
Oblivious sorting

• Comparison-based sorting, but fix the number of comparisons

• Also called a *sorting network*

• Batcher’s algorithm
  • Sort the first half of a list, and sort the second half of that list
  • Sort the odd-indexed values, then even-indexed values
  • One more comparison-switch per pair of keys
  • Proof of security?
Batcher’s algorithm correctness

- **Theorem:** Batcher’s algorithm described on the previous slide results in a sorted list.

- **Proof:** Let the list’s size be $n$ where $n$ is a multiple of 4. Denote the list as $X$. If two halves have been sorted separately, then for all elements between 1 and $n$ except for $X_1$ and $X_{n/2+1}$, $X_{i-1} \leq X_i$. We call $X_{i-1}$ the predecessor of $X_i$.

  Both 1 and $n/2 + 1$ are odd, so the above is true for every even-indexed value, and true for every odd value except for two.
Batcher’s algorithm correctness

- **Theorem:** Batcher’s algorithm described on the previous slide results in a sorted list.

- **Proof (cont’d):** Let $Y_{even}$ denote the sorted even values, and $Y_{odd}$ denote the sorted odd values.

  - $Y_{even,l}$ must be larger than at least $l$ values in $Y_{odd}$.
  - $Y_{odd,l+1}$ must be larger than at least $l - 1$ values in $Y_{even}$. 
Batcher’s algorithm correctness

- **Theorem:** Batcher’s algorithm described on the previous slide results in a sorted list.

- **Proof (cont’d):** Let \( Y \) be the list after the 4 sorts. Let \( l \in \{1, \ldots, n/2\} \). This means that

\[
Y_{2l-1} \leq Y_{2l} \quad \text{and} \quad Y_{2l-2} \leq Y_{2l+1} \quad \text{(via last slide’s argument)}
\]

\[
Y_{2l-2} \leq Y_{2l} \quad \text{and} \quad Y_{2l-1} \leq Y_{2l+1} \quad \text{(because even and odd values are sorted separately)}
\]

Therefore, the elements in pairs \((Y_{2l}, Y_{2l+1})\) are ordered with respect to adjacent pairs. So the final step is to sort within these pairs!
Batcher’s algorithm

- Applied recursively until there are only two elements, where sort = comparison
- Another optimization: after the two halves are sorted, the odd and even indexed values are partially sorted, so needs a merge instead of a full sort
Today’s readings:
oblivious analytics in SGX